Najdaljše zaporedje besed

Izziv je bil torej tak: imamo seznam besed. Sestaviti želimo najdaljšo verigo besed, v kateri se vsaka beseda začne z zadnjo črko prejšnje. Besede se ne smejo ponavljati. Veriga se lahko začne s poljubno besedo.

Pri reševanju lahko uporabljamo nekoliko krajši slovar besed in enega z več kot 1500 besedami, ki jih lahko uvozimo iz modula `besede`, ki je priložen nalogi.

```
slovar = ["ABRAHAM", "MELODIJA", "ASTEROID", "DREVO", "MEČ", "OBLAK", "KLEPTOMAN", "KAČA"]
from besede import nouns
```

Izčrpno preiskovanje

Očitno je prva možnost, da pregledamo vsa možna zaporedja. Enako očitno je tudi, da je teh zaporedij veliko.

Nekateri študenti so se lotili tega izziva. To je pohvalno.

```
def vse_verige(slovar):
    verige = []
    slovar = set(slovar)
    for prva in slovar:
        verige += nadaljevanja(prva, slovar - {prva})
    return verige

def nadaljevanja(beseda, slovar):
    rezultat = [[beseda]]
    for naslednja in slovar:
        if naslednja[0] == beseda[-1]:
            for podverige in nadaljevanja(naslednja, slovar - {naslednja}):
                rezultat.append([beseda] + podverige)
    return rezultat

vse_verige(slovar)
```

```
[['KAČA'],
 ['KAČA', 'ASTEROID'],
 ['KAČA', 'ASTEROID', 'DREVO'],
 ['KAČA', 'ASTEROID', 'DREVO', 'OBLAK'],
 ['KAČA', 'ASTEROID', 'DREVO', 'OBLAK', 'KLEPTOMAN'],
 ['KAČA', 'ABRAHAM'],
 ['KAČA', 'ABRAHAM', 'MEČ'],
 ['KAČA', 'ABRAHAM', 'MELODIJA'],
 ['KAČA', 'ABRAHAM', 'MELODIJA', 'ASTEROID'],
 ['KAČA', 'ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO'],
 ['KAČA', 'ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK'],
 ['KAČA', 'ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KLEPTOMAN'],
 ['DREVO'],
```

```
['DREVO', 'OBLAK'],
['DREVO', 'OBLAK', 'KAČA'],
['DREVO', 'OBLAK', 'KAČA', 'ASTEROID'],
['DREVO', 'OBLAK', 'KAČA', 'ABRAHAM'],
['DREVO', 'OBLAK', 'KAČA', 'ABRAHAM', 'MEČ'],
['DREVO', 'OBLAK', 'KAČA', 'ABRAHAM', 'MELODIJA'],
['DREVO', 'OBLAK', 'KAČA', 'ABRAHAM', 'MELODIJA', 'ASTEROID'],
['DREVO', 'OBLAK', 'KLEPTOMAN'],
['ASTEROID'],
['ASTEROID', 'DREVO'],
['ASTEROID', 'DREVO', 'OBLAK'],
['ASTEROID', 'DREVO', 'OBLAK', 'KAČA'],
['ASTEROID', 'DREVO', 'OBLAK', 'KAČA', 'ABRAHAM'],
['ASTEROID', 'DREVO', 'OBLAK', 'KAČA', 'ABRAHAM', 'MEČ'],
['ASTEROID', 'DREVO', 'OBLAK', 'KAČA', 'ABRAHAM', 'MELODIJA'],
['ASTEROID', 'DREVO', 'OBLAK', 'KLEPTOMAN'],
['MEČ'],
['MELODIJA'],
['MELODIJA', 'ASTEROID'],
['MELODIJA', 'ASTEROID', 'DREVO'],
['MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK'],
['MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KAČA'],
['MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KAČA', 'ABRAHAM'],
['MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KAČA', 'ABRAHAM', 'MEČ'],
['MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KLEPTOMAN'],
['MELODIJA', 'ABRAHAM'],
['MELODIJA', 'ABRAHAM', 'MEČ'],
['OBLAK'],
['OBLAK', 'KAČA'],
['OBLAK', 'KAČA', 'ASTEROID'],
['OBLAK', 'KAČA', 'ASTEROID', 'DREVO'],
['OBLAK', 'KAČA', 'ABRAHAM'],
['OBLAK', 'KAČA', 'ABRAHAM', 'MEČ'],
['OBLAK', 'KAČA', 'ABRAHAM', 'MELODIJA'],
['OBLAK', 'KAČA', 'ABRAHAM', 'MELODIJA', 'ASTEROID'],
['OBLAK', 'KAČA', 'ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO'],
['OBLAK', 'KLEPTOMAN'],
['KLEPTOMAN'],
['ABRAHAM'],
['ABRAHAM', 'MEČ'],
['ABRAHAM', 'MELODIJA'],
['ABRAHAM', 'MELODIJA', 'ASTEROID'],
['ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO'],
['ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK'],
['ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KAČA'],
['ABRAHAM', 'MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KLEPTOMAN']]
```

Prva funkcija ni bila nič posebnega.

Mali trik: `slovar` pretvori v množico. Na ta način lahko preprosteje odstranjujemo uporabljene besede. O kakšni posebni hitrosti pa pri teh funkcijah itak ne moremo govoriti.

Funkcija `vse_verige` gre čez vse možne prve besede in v seznam verig (`verige`) doda vse verige, ki jih vrne funkcija `nadaljevanja`, ki, kot pove njeno ime, poišče vsa možna nadaljevanja pri podani besedi in množici še neuporabljenih besed.

```
def vse_verige(slovar):
    verige = []
    slovar = set(slovar)
    for prva in slovar:
        verige += nadaljevanja(prva, slovar - {prva})
    return verige
```

Druga funkcija je zanimivejša. V seznam `rezultat` nabira vse možne verige, ki se začnejo s podano besedo. Za začetek je to veriga, ki vsebuje samo to besedo, torej veriga `[beseda]`. Ker gre za seznam verig, je to v začetku seznam, ki vsebuje, `[beseda]`, se pravi `[[beseda]]`.

Nato gre funkcija čez vse besede, ki so še v slovarju. Če se beseeda začne z ustrezno črko, gre z zanko `for` čez vsa možna nadaljevanja, ki se začnejo s to, naslednjo besedo, se pravi `for podverige in nadaljevanja(naslednja, slovar - {naslednja})`. Vsako tako "podverigo" dodajo v seznam, na začetek pa pripnejo prvo besedo (`beseda`).

```
def nadaljevanja(beseda, slovar):
    rezultat = [[beseda]]
    for naslednja in slovar:
        if naslednja[0] == beseda[-1]:
            for podverige in nadaljevanja(naslednja, slovar - {naslednja}):
                rezultat.append([beseda] + podverige)
    return rezultat
```

To je to.

Seznam je dolg že pri osmih besedah. Besed v seznamu, podanem ob domači nalogi, je bilo približno 1500, zato ta rešitev tam ne bo delovala - ne glede na to, kako jo sfriziramo. Po mojem.

Preden napišemo rešitev, ki deluje tudi za daljše sezname, povejmo le, da se da obe funkciji združiti v eno, vendar s tem ne pridobimo ničesar bistvenega. Tako, kot je, je lepše. Poleg tega povejmo še, da se ju da zapisati kompaktneje.

```
def najdaljsa_veriga(slovar):
    return max(sum((nadaljevanja(prva, set(slovar)) for prva in slovar), []), key=len)

def nadaljevanja(beseda, slovar):
```

```
    return [[beseda] + pod
            for naprej in slovar if naprej[0] == beseda[-1]
            for pod in nadaljevanja(naprej, slovar - {beseda})] or [[beseda]]

najdaljsa_veriga(slovar)
```

```
['MELODIJA', 'ASTEROID', 'DREVO', 'OBLAK', 'KAČA', 'ABRAHAM', 'MEČ']
```

Ali je to boljše, je stvar okusa. Najbrž slabega okusa.

(Mimogrede lahko ugotovimo tudi, da je to res najdaljše, kar je možno sestaviti iz `["ABRAHAM", "MELODIJA", "ASTEROID", "DREVO", "MEČ", "OBLAK", "KLEPTOMAN", "KAČA"]`. Imamo dve besedi, ki se začneta s K in le eno, ki so konča s K. Poleg tega je ena od besed na K, KLEPTOMAN, lahko le zadnja, saj ni besed na N. Pred KLEPTOMAN je lahko le OBLAK, pred OBLAK le DREVO, pred DREVO ASTEROID, pred ASTEROID pa KAČA ali MELODIJA. Takšno zaporedje, MELODIJA, ASTEROID, DREVO, OBLAK, KLEPTOMAN ima pet besed. Besede KLEPTOMAN torej en bo v rešitvi. Najdena rešitev pa vsebuje vse besede razen KLEPTOMAN, torej je najdaljša možna.)

Približna rešitev

Kadar ne moremo najti najboljše možne rešitve, bomo morali biti zadovoljni s kar dobro. Tudi šahist ne more razmisliti vseh možnih nadaljevanj igre, temveč razmišlja za nekaj potez naprej. Nekaj podobnega bomo naredili tu: med možnimi nadaljevanji bomo izbrali tistega, ki se konča s črko, na katero se začne čimveč besed. Na ta način se izognemo besedam, ki porabljajo redke, dragocene črke.

Najprej pomislimo, da se splača razdeliti besede glede na črko, s katero se začenjajo.

```
from collections import defaultdict

def razdeli_besede(slovar):
    po_zacetnicah = defaultdict(list)
    for beseda in slovar:
        po_zacetnicah[beseda[0]].append(beseda)
    return po_zacetnicah

po_zacetnicah = razdeli_besede(nouns)
```

Zaradi praktičnosti smo uporabili `defaultdict`. Ne o tem - pa tudi o slovarjih - se še nismo učili, zato se tudi zdaj ne bomo poglabljali v to. Vedimo le: z, recimo, `po_zacetnicah["K"]` dobimo vse besede, ki se začnejo s črko K.

```
print(po_zacetnicah["K"])
```

```
['KNOWLEDGE', 'KING', 'KEY', 'KIND', 'KITCHEN', 'KID', 'KNIFE', 'KNEE', 'KEEP', 'KICK', 'KI
```

Program bomo sestavili iz dveh funkcij. Prva bo iz seznama možnih nadaljevanj izbrala eno (poskusili bomo: čim boljše) nadaljevanje. Druga bo klicala prvo.

```
def najboljsa_naslednja(moznosti, po_zacetnicah):
    if not moznosti:
        return None
    return moznosti[0]

def sestavi_zaporedje(slovar):
    po_zacetnicah = razdeli_besede(slovar)
    naslednja = najboljsa_naslednja(slovar, po_zacetnicah)
    zaporedje = []
    while naslednja:
        zaporedje.append(naslednja)
        po_zacetnicah[naslednja[0]].remove(naslednja)
        naslednja = najboljsa_naslednja(po_zacetnicah[naslednja[-1]], po_zacetnicah)
    return zaporedje
```

Naš prvi približek `najboljsa_naslednja` je neumen: vrne preprosto prvo možnost. Če je seznam možnosti prazen, vrne `None`. Funkcija sicer dobi tudi slovar `po_zacetnicah`, vendar ga - v tej, neumni obliki - ne uporablja.

Funkcija `sestavi_zaporedje` prejme slovar vseh besed. Razbije ga na seznam po začetnicah. Izbere prvo besedo (ki ji iz praktičnih razlogov, reče `naslednja` in pripravi prazno zaporedje.

Nato ponavlja, dokler ima kako naslednjo besedo: besedo doda v zaporedje, jo pobriše iz seznama vseh besed, ki se začnejo s to črko in nato izbere naslednjo besedo, tako da pokliče prvo funkcijo.

Ugotovimo, kako dobro to deluje.

```
zaporedje = sestavi_zaporedje(nouns)
print(zaporedje)
print(len(zaporedje))
```

```
['PEOPLE', 'ECONOMICS', 'SYSTEM', 'MAP', 'PERSON', 'NATURE', 'EXAM', 'MEAT', 'TWO', 'OVEN',
65
```

To ni tako slabo, vendar bo boljše, če naredimo kakšno manj neumno izbiro naslednje besede. Gotovo se nam splača varčevati z redkimi besedami: za naslednjo besedo nočemo izbrati besede, ki se konča s črko, s katero se začenja malo besed. Oziroma obratno: izmed različnih možnosti bomo za naslednjo besedo izbrali takšno, s katero se začenja čim več besed.

```
def najboljsa_naslednja(moznosti, po_crkah):
    naj_naprej = None
    naj_pogostost = -1
    for naprej in moznosti:
        pogostost = len(po_crkah[naprej[-1]])
```

```
        if pogostost > naj_pogostost:
            naj_pogostost = pogostost
            naj_naprej = naprej
    return naj_naprej

def sestavi_zaporedje(slovar):
    po_zacetnicah = razdeli_besede(slovar)
    naslednja = najboljsa_naslednja(slovar, po_zacetnicah)
    zaporedje = []
    while naslednja:
        zaporedje.append(naslednja)
        po_zacetnicah[naslednja[0]].remove(naslednja)
        naslednja = najboljsa_naslednja(po_zacetnicah[naslednja[-1]], po_zacetnicah)
    return zaporedje

zaporedje = sestavi_zaporedje(nouns)
print(len(zaporedje))
```

```
523
```